Real-Time
Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Antialiasing

Outline

- What are aliasing and antialiasing?
- Taxonomy of antialiasing approaches
- Exploration of details

Readings

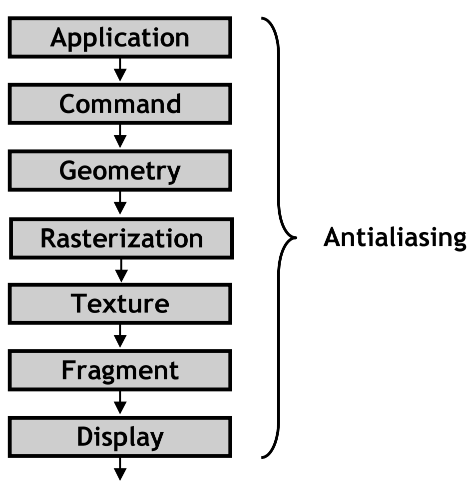
Required

- *Filtering Edges for Gray-Scale Displays*, Gupta and Sproull, SIGGRAPH Proceedings ’81.
- *A Parallel Algorithm for Polygon Rasterization*, Pineda, Computer Graphics 22, 4 (August ’88).
- *A New Simple and Efficient Antialiasing with Subpixel Masks*, Andreas Schilling, SIGGRAPH ’91.

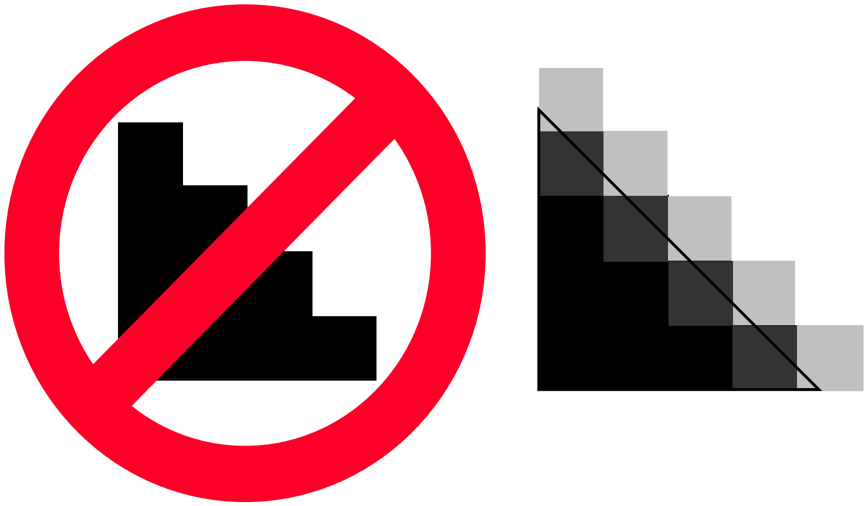
Recommended

- Multisample extension to OpenGL.

Modern Graphics Pipeline



What is "Antialiasing"?



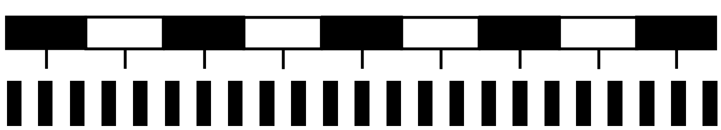
What is "Aliasing"?

Result of sampling below the Nyquist rate?

- But geometric input has energy at all frequencies
- And there's no practical way to change this

Reconstruction of a strong low-frequency "alias" of an input signal component above the Nyquist limit?

- Agrees with common understanding in signal processing terms,
- But still doesn't cover the "jaggies" case



Radical Thought

Maybe signal processing theory isn't the best way to approach the problem of eliminating jaggies.

- Can't band-limit the geometric input
- Jaggies typically aren't aliasing anyway
- Image is constructed in the framebuffer, not just filtered there.

So how should we think of this problem?

Ideal Jaggie Removal - Integration

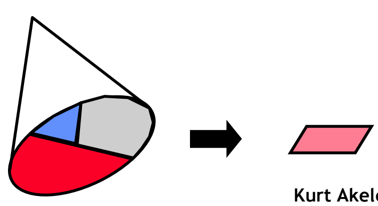
Define a 2D spatial filter function for a pixel

- Probably not a box filter (though may be)
- Probably not Sync function (infinite extent is unworkable)
- Empirical, depends on display properties

Render image into an infinite-precision shapes buffer

- Hidden geometry is somehow eliminated,
- Leaving exact geometry and color information

Integrate filter function with geometry/color info for each pixel



Antialiasing System Goals

Best static image

Good dynamic image

- Avoid sudden frame-to-frame changes
  - Good model: bilinear interpolation in texture filtering
  - Avoid negative-training (e.g. pulsing aircraft on horizon)

Reasonable

- Hardware and performance costs
- Implementation and application complexity

Integration with other GPU features

- Depth buffer for occlusion computation
- Stencil buffer
- Transparency?

Taxonomy of Antialiasing Methods

Two fundamental approaches, based on what coverage info is

- Computed per fragment, and
- Stored per pixel in the framebuffer
- Note: coverage may be pre-integrated with filter function

Fractional

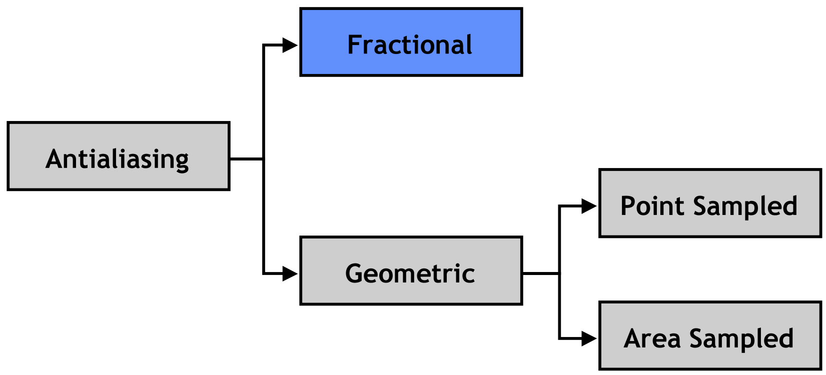
- No geometric information
- OpenGL "smooth" antialiasing

Geometric

- Some geometric information
  - Point sampling
  - Area sampling
- OpenGL "multisample" antialiasing

Each approach has strengths and weaknesses

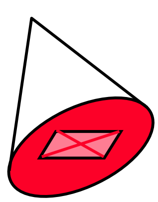
Taxonomy



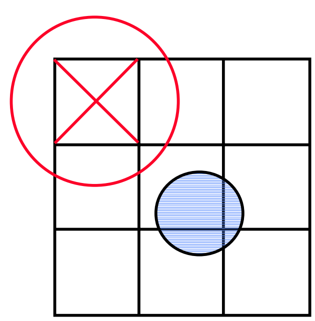
Fractional Antialiased Points

Compute percent coverage by integrating:

- Point “geometry”
- With each pixel filter function that intersects the point geometry



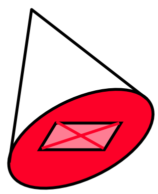
Pixel filter function



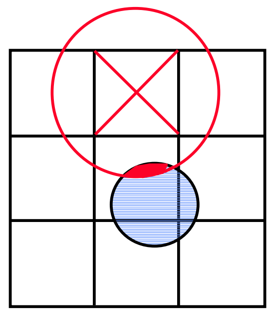
Fractional Antialiased Points

Compute percent coverage by integrating:

- Point "geometry"
- With each pixel filter function that intersects the point geometry



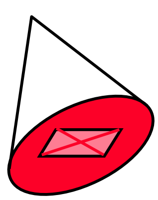
Pixel filter function



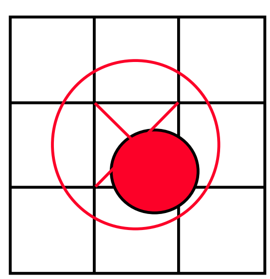
Fractional Antialiased Points

Compute percent coverage by integrating:

- Point "geometry"
- With each pixel filter function that intersects the point geometry



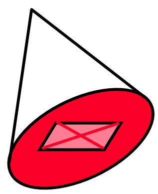
Pixel filter function



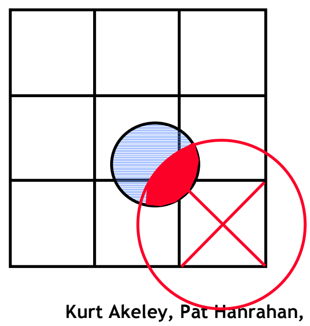
Fractional Antialiased Points

Compute percent coverage by integrating:

- Point "geometry"
- With each pixel filter function that intersects the point geometry



Pixel filter function



Hardware Implementation

For each point size and sub-pixel point location

- Pre-convolve for each affected pixel
- Store results in a hardware table

Reduce table size by

- Limiting the number of supported point sizes
  - Reduces table outputs too
- Limiting sub-pixel position resolution
- Exploiting symmetry
  - Horizontal
  - Vertical
  - Diagonal

Optional: normalize aggregate point intensity (coverage)

- Avoid frame-to-frame strobe effects for moving points

Framebuffer Operations

Point on background

- Blend point color with background color
- Use coverage at each pixel to determine blend

$$C'_{fb} = A_f\, C_f + (1 - A_f)\, C_{fb} \qquad A_f \textit{ is coverage}$$

Point intersecting point

- Geometric relationship is unknown
- Best guess → random
- Use same blend function!
  - Call this blend function "uncorrelated"

Works recursively for all points

Fractional Antialiased Points

Strengths

- Excellent static and dynamic image quality
  - Point overlaps are stable if not accurate
  - Strobing effects are eliminated by aggregate intensity normalization
- Simple and inexpensive to implement and use
  - Framebuffer gets blend function, no added storage
- Operates with depth and stencil buffers

Weaknesses

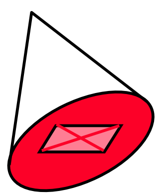
- Depth buffer yields non-optimal results
  - Nearer small coverage replaces farther large coverage

Fractional Antialiased Lines

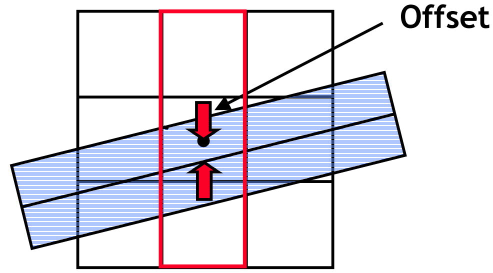
Table is larger

- Line width, offset to pixel center, slope
- Must exploit symmetry for reasonable table size
- End-of-line filtering can be very expensive

Iterate 1 x *n* function



Pixel filter function

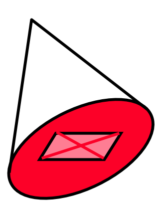


Fractional Antialiased Lines

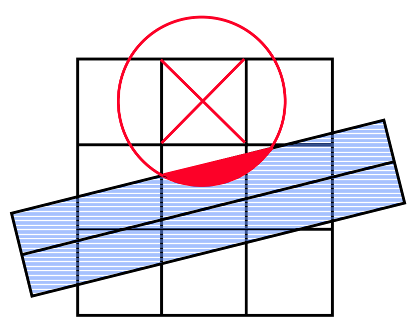
Table is larger

- Line width, offset to pixel center, slope
- Must exploit symmetry for reasonable table size
- End-of-line filtering can be very expensive

Iterate 1 x *n* function



Pixel filter function

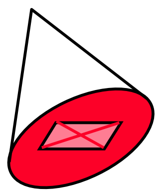


Fractional Antialiased Lines

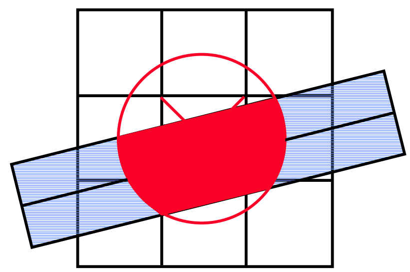
Table is larger

- Line width, offset to pixel center, slope
- Must exploit symmetry for reasonable table size
- End-of-line filtering can be very expensive

Iterate 1 x *n* function



Pixel filter function

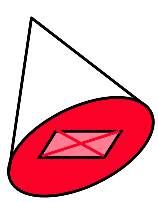


Fractional Antialiased Lines

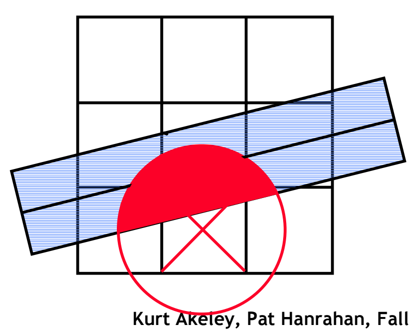
Table is larger

- Line width, offset to pixel center, slope
- Must exploit symmetry for reasonable table size
- End-of-line filtering can be very expensive

Iterate 1 x *n* function



Pixel filter function



Fractional Antialiased Lines

Strengths

- Very good static and dynamic image quality
  - Line overlaps are stable if not accurate
  - Roping effects are eliminated by aggregate intensity normalization
- Simple and inexpensive to implement and use
  - Framebuffer gets blend function, no added storage
- (Barely) operates with depth and stencil buffers

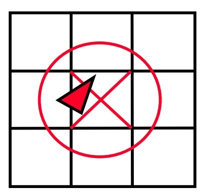
Weaknesses

- Depth buffer yields very non-optimal results
  - Nearer small coverage replaces farther large coverage
  - Depthcue colors interact badly

Fractional Antialiased Triangles

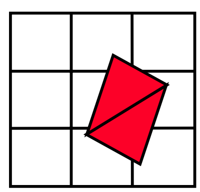
Difficult to pre-compute coverage integrations

- Edge slopes OK, but
- Vertexes introduce two edge slopes, and
- Small triangles have all 3 edges in play!



Blending approximation breaks down completely

- Uncorrelated blend leaves visible seams
- Adjacent triangles are anti-correlated, not uncorrelated



Anti-correlated Blend Functions

Assuming perfect tiling, depth complexity 1.0

- E.g. 2D rendering (a clock face, for example)

$$C'_{fb} = A_f \, C_f + C_{fb}$$

Assuming nearest-to-farthest primitive sorting

- Special case of 3D rendering
- Requires addition of alpha channel in framebuffer

$$i = min(A_f, (1 - A_{fb}))$$

$$A'_{fb} = A_{fb} + i$$

$$C'_{fb} = i \, C_f + C_{fb}$$

Fractional Antialiased Triangles

Strengths

- Produces useful results in very specialized circumstances
- Requires minimal framebuffer additions
  - Anti-correlated (saturation) blend, alpha buffer

Weaknesses

- Filter quality is poor
  - Table is impossible to implement, so
  - Convolution is typically with a box filter
- Difficult and expensive to implement
- Fails entirely with depth buffer, stencil

Fractional Antialiasing Summary

Great for single-colored dot clouds

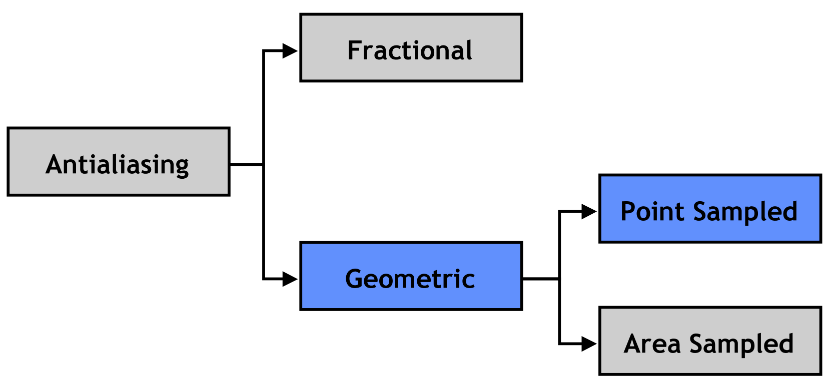
Good for lines

- High-quality filtering, but
- Problems with line-line intersections

Almost useless for triangles

- Expensive to implement
- Filtering quality is poor
- Depth buffer fails completely

Taxonomy



Multi-pass Accumulation Buffer AA

Advantages

- Logical performance/quality ratio
- Simple to implement and to use (e.g. depth buffer)
- Point sample pattern is arbitrary
- "Free" anisotropic texture filtering ….

Disadvantages

- Shading is too expensive
  - Reyes renderer shades just once or twice per pixel
  - Perception: NTSC chroma vs. luminance bandwidth
- Computation and bandwidth are replicated
  - Application, Command, Geometry

Transistors are cheap, communication is expensive

Multisample Antialiasing

Specify the location of multiple sample points per pixel

- Patterns may differ spatially, but not temporally

Rasterize fragments that include

- A bitmask of occluded samples
- Appropriate color, depth, and texture coords

Evaluate texture once per fragment (not per sample)

Store color and depth for each sample in framebuffer

Resolve samples to final pixel value either

- Each time the pixel is modified, or
- Once, before the buffer is displayed

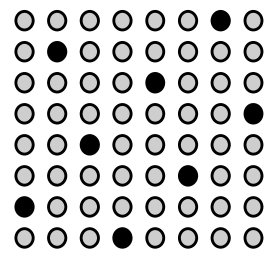
Multisample Sample Pattern

Trade-off

- Pseudo random
  - Better, more efficient filtering
- Regular
  - Easier, more efficient rasterization

Compromise pattern is regular subset

- Manageable sample count
- Empirically best
- Pixar owns patent on this



Rasterization Fragment Selection

Box sampled, as in tiled rasterization

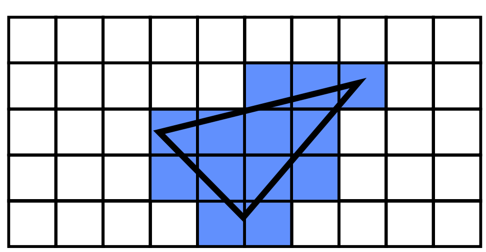
The bitmask is composed of point samples

Pixel's box must enclose all sample locations

Might be outside the 1 x 1 ideal pixel area

Look how pixel depth complexity has increased!

Area is 5.0, but 13 fragments are generated



Rasterization Parameter Assignment

Sample depth at each occluded sample location

- Depth buffer controls "geometry" of final image

Sample color once per fragment

- Do not sample outside the triangle!
- Choose a sample location in a repeatable manner
  - Occluded sample nearest to pixel center
  - Occluded sample nearest to "fragment" center

Sample texture coordinates once per fragment

- Pixel center – optimizes for adjacent triangles
- Color sample location – optimizes for silhouette

Multisample Framebuffer

Store full depth and color values for each sample

Execute full fragment operations for each sample

- Depth buffer
- Stencil buffer
- Blending
- …

Resolve to final color

- Only final color buffers are double buffered

Examples: high-end SGI machines

8-Sample Multisample Framebuffer

```
typedef struct {
    int red, green, blue, alpha;
} Color;

typedef struct {
    Color c;
    int depth;
    int stencil;
} Sample;

typedef struct {
    Color front, back;
    Sample s[8];
} Pixel;
```


Ideal Multisample Summary

Strengths

- Good full-scene antialiasing quality
- Works seamlessly with depth/stencil buffers
  - Even works for interpenetrating triangles
- Operation is predictable and reliable

Weaknesses

- Framebuffer is very expensive
  - Bandwidth
  - Memory requirements
- Point and line filter quality is mediocre
  - Fractional approach has higher filter quality
  - But multisample correctly resolves intersections

Merged Multisample and Fractional

OpenGL Multisample spec designed to allow this

Enable multisample framebuffer

Render triangles in multisample mode

Then render points and lines in fractional mode

- OpenGL "smooth"

Result is

- High-quality points and lines
- Good quality filtering of solids
  - No seams or cracks
- Proper occlusion point/line/solid to solid
  - Point/line to point/line occlusion still poor

Architectural Options

Store only mask in framebuffer

- Assume front-to-back rendering
- Intersect and accumulate masks to determine blend function
- Examples: early flight simulation IGs

$i = (M_f \text{ AND } \overline{M_{fb}}) / n$

$M'_{fb} = M_{fb} \text{ OR } M_f$

$C'_{fb} = i\, C_f + C_{fb}$

Mask Rendering

$i = \min(A_f, (1 - A_{fb}))$

$A'_{fb} = A_{fb} + i$

$C'_{fb} = i\, C_f + C_{fb}$

Anti-correlated Rendering

Architectural Options (continued)

Tiled rendering

- Reduces framebuffer storage requirements
  - Multisample buffers needed only for active tiles
- But requires region binned geometry
  - Extra frame of latency
  - Lots of data management complexity
- Examples
  - Pixel Planes 5, PixelFlow
  - Talisman
  - GigaPixel, ….

Architectural Options (continued)

Loss-less compression

- One and two-fragment special cases
  - Saves bandwidth, not memory

Almost-loss-less compression

- Reduced color precision per sample
  - E.g. 16 8-bit samples reconstruct a 12-bit color value
  - Saves bandwidth and memory

Lossy compression at pixels

- Examples: endless research papers
- Watch out for failure modes!

System Evaluation

Simulate possible algorithms

- Run test patterns
- Run scenes from real applications
- Try to break it – application developers will!

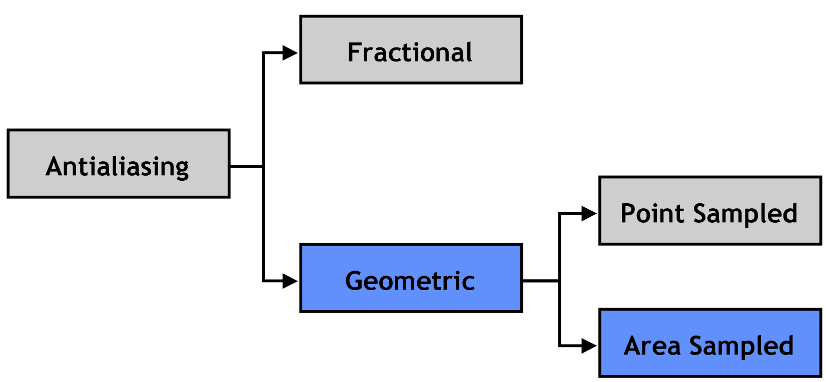
Study individual images carefully

More important: study sequences of images!

- Static images do not tell the whole story

Document evaluation procedure when publishing

Taxonomy



Area Sampling

How can one get the good qualities of point sampling

- Accurate depth comparisons
- Samples taken only within triangle boundaries
- "Perfect" anti-correlated blending
- Robust, reliable algorithms

With the greater filter quality of area-based sampling

- Greater spatial resolution
- High-quality, display-tuned filter function

In one algorithm?

I know of no general solution

But there are "hacks" that get some value

2-fragment Area Filtering

Optimize for special case of just two visible fragments

For each fragment compute

- Coverage mask
- Filter-function-integrated coverage value

At each pixel store

- All multisample values
- One coverage value, and
- One extra state bit (tracks 2-fragment case)

When merging fragment colors during resolution

- Use coverage value in 2-fragment case
- Use multisample values otherwise

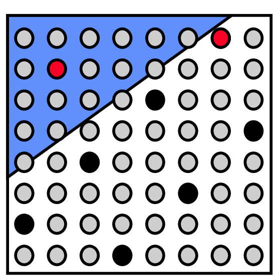
"Schilling" Antialiasing

Choose samples based on coverage, not just geometry

Good idea, but lots of problems

- Must sample outside primitive
  - Colors wrap
  - T-junctions protrude

Blue triangle occludes ¼ of the pixel's unit area, but only one sample. Select a second sample to get the best "weight"



A-buffer and Relatives

Variable data structure at each pixel

Strengths

- Handles transparency and depth occlusion
- Simple for applications to program

Weaknesses

- Complex and fragile to implement
  - Failure modes are unpleasant!

Hardware Workshop Papers

R-Buffer: A Pointerless A-Buffer Hardware Architecture, Wittenbrink, Graphics Hardware 2001.

Single-Pass Full-Screen Hardware Accelerated Antialiasing, Lee and Kim, 2000.

Prefiltered Antialiased Lines Using Half-Plane Distance Functions, McNamara, McCormack, Jouppi, 2000.

Z3: An Economical Hardware Technique for High-Quality Antialiasing and Transparency, Jouppi, Chang, 1999.

High-Quality Rendering Using the Talisman Architecture, Barkans, 1997

Real-Time
Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall